Block or Convolutional AL-FEC Codes? A Performance Comparison for Robust Low-Latency Communications

Vincent Roca* Belkacem Teibi* Christophe Burdinat † Tuan Tran-Thai † Cédric Thienot †
*Inria, France †Expway, France
{vincent.roca,belkacem.teibi}@inria.fr, {christophe.burdinat,tuan.tran,cedric.thienot}@expway.com

***Abstract*—Application-Level Forward Erasure Correction (AL-FEC) codes are a key element of telecommunication systems. They are used to recover from packet losses during large scale content distribution, for instance within the FLUTE/ALC (file transfers) and FECFRAME (continuous real-time media transfers) protocols of the 3GPP Multimedia Broadcast and Multicast Services (MBMS) standard. However currently standardized and deployed AL-FEC codes for these protocols (e.g., Raptor(Q) or LDPC-Staircase) are all block codes which means that the data flow must be segmented into blocks of predefined size. Surprisingly AL-FEC codes based on a sliding encoding window have not yet been considered in spite of their major advantages.**

This work analyzes both types of codes in the context of real-time (e.g., multimedia) flows. More precisely, it details how to initialize block and convolutional AL-FEC codes to comply with real-time constraints and introduces the "decoding beyond maximum latency" optimization to convolutional codes. Then it compares the added FEC-related latency of both solutions and the decoding throughput of the two codecs. This work highlights the major benefits of convolutional codes for the large scale distribution of real-time flows and supports the idea of extending FECFRAME specifications (RFC 6363) to support convolutional FEC codes.

I. Introduction

Internet and wireless networks can be regarded as erasure channels, where router congestions, severe packet corruptions that cannot be recovered (e.g., caused by poor reception conditions), or intermittent connectivity are source of packet losses (or "erasures"). If retransmissions can improve robustness (e.g., with TCP), there are situations where it is not practical: the return channel required by feedbacks does not exist with a unidirectional broadcast network, scalability issues may prevent the use of feedbacks (e.g., broadcast sessions to a huge number of receivers), and the extra Round Trip Time (RTT) delay generated by feedback and retransmission may be too large for real-time flows.

This is why Application-Level Forward Erasure Correction (AL-FEC) codes have become a key component of large scale content distribution systems, relying on broadcast/multicast technologies to efficiently send the same content to a huge number of receivers. This is the case for large scale file content delivery, where scalability does matter even if latency is not a major constraint (e.g., the same content can be made available for hours or days, leaving the opportunity for interested users to join the session at their discretion, recover the content and leave). This is also the case for large scale real-time media transfer applications, where in addition to scalability, latency does matter. A typical example is a popular sport event where multiple video streams from various cameras in the stadium are broadcast in real time to the spectators's smartphones/tablets, using 4G or Wifi multicast capabilities. The first use-case is typically managed by the **FLUTE/ALC** protocol stack, standardized by IETF as RFC 6726 [1], while the second use-case is managed by the **FECFRAME** protocol, standardized by IETF as RFC 6363 [2]. Both protocols are also part of the 3GPP Multimedia Broadcast and Multicast Services (MBMS) standard [3], meaning they will soon be deployed (it is already the case for FLUTE/ALC in certain countries).

However currently standardized AL-FEC codes for FLUTE/ALC and FECFRAME (XOR-based, Raptor(Q) [4], [5], Reed-Solomon [6] and LDPC-Staircase [7]) are block codes: they require the data flow to be first segmented into blocks of predefined size. A major limitation is the added uncompressed latency, caused by the AL-FEC encoding and decoding process, which is penalizing with real-time flows. Surprisingly AL-FEC codes based on a sliding encoding window, also called convolutional codes, have not yet been considered for these protocols.

In this work we compare block and convolutional AL-FEC codes for real-time broadcast and multicast distribution (e.g., using FECFRAME) and analyze their performance. We choose Reed-Solomon as a representative of ideal, MDS block codes, and Random Linear Codes over GF(2^8) (RLC) as representative of convolutional codes. The contributions are threefold:

- it explains how to initialize the internal parameters of block and convolutional AL-FEC codes in order to comply with real-time constraints;
- it introduces the "decoding beyond maximum latency" optimization that improves the erasure recovery performance of convolutional codes;
- finally it highlights the major benefits of convolutional codes. It shows that receivers experiencing good to medium channel quality have an extra FEC-related latency close to zero. Only very bad receivers experience an extra latency that approaches that of block codes while keeping similar erasure recovery performance. This is a major benefit compared to block codes where all receivers experience the same latency, no matter their reception quality.

II. Problem Position for Robust Real-Time Flows

Let us first define the problem by considering a real-time flow with strict timing constraints. This source flow is composed of Application Data Units (ADUs) (we use FECFRAME's terminology [2]) coming from the sending application. ADUs are typically RTP packets containing audio and/or video content. Since each ADU features a maximum validity period, it must be delivered to the receiving application before it expires, which creates constraints on the full transmission chain, from the sender to the receiver. In practice ADUs are potentially of variable size and may contribute to one or more source symbols each. Additionally, FECFRAME prepends a two-byte "ADU length" field and a one-byte "flow identifier" field (if several flows are protected together) to each ADU, and adds padding such that each augmented ADU size be a multiple of the symbol size.

In order to simplify the analysis **we assume that ADUs are all of the same size and contribute to exactly one source symbol**. We also assume, without loss of generality, that **the network features a constant transmission delay (i.e., there is no jitter) and we ignore UDP/IP processing times**. Therefore **we only focus on the AL-FEC related latency**, both during encoding at the sender and decoding at a receiver.

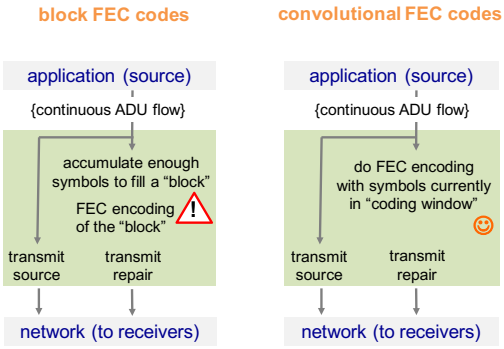


Fig. 1. Block versus convolutional AL-FEC coding principles at a sender.

A. The Case of Block AL-FEC Codes

With block AL-FEC codes, the input ADU flow is segmented into a sequence of blocks of appropriate size, and FEC encoding (at a sender) and decoding (at a receiver) are performed independently on a per-block basis. Independently of the nature of the code, this directly impacts latency: even if ADUs (and source symbols) are immediately sent, repair symbols are delayed by the block creation time, which directly depends on the number k of source symbols. This block creation time directly impacts the receivers: no erasure recovery can occur before this delay since repair symbols are sent after.

A good value for the block size is necessarily a balance between the maximum decoding latency at the receivers (which increases with k) and the desired robustness in front of long erasure bursts (which also increases with k). In practice, the block size k is set to the maximum value made possible by real-time flow requirements, which provides maximum robustness in front of erasure bursts while staying within delay bounds. A downside is that all receivers, even those experiencing excellent transmission conditions, are constrained and penalized by this decoding latency.

B. The Case of Convolutional AL-FEC Codes

On the opposite, an AL-FEC code based on a sliding encoding window (of fixed or elastic size) can remove most of this decoding delay. Indeed, repair symbols are generated and sent on-the-fly, at any time, from the set of source symbols already present in the current encoding window. At the receiver, an erased source symbol can therefore be recovered thanks to the next repair symbol received if the linear system rank permits it. The FEC-related decoding latency essentially depends on the channel erasure loss and applied coding rate, and is usually rather low for most receivers as we will see. Using these codes is therefore highly beneficial.

III. Code Configuration with Real-Time Flows

A. Added Latency Evaluation Methodology

Let us now discuss the optimal configuration of the two types of AL-FEC codes. We first make a few assumptions:

- the application generates fixed size ADUs that contribute to exactly one source symbol each, of size, $symb\_size$ bytes;
- the application generates a Constant BitRate (CBR) flow, equal to bw bits per second, which is also the ADU bitrate over the network. We do not consider the repair packet bitrate in this work;
- the packet transmission time is assumed constant (i.e., no jitter), as well as the UDP/IP datagram processing times. Being out of scope, they are ignored;
- ADUs are returned to the receiving application as soon as possible, upon reception or decoding;

We also add two definitions:

- each ADU of the application flow has a maximum end-to-end latency constraint, from which we can safely remove the constant communication delay (see above). The resulting transport protocol (e.g., within FECFRAME) maximum latency that cumulates the sender plus receiver processing times is equal to $tp\_max\_lat\_in\_sec$ seconds;
- the AL-FEC code rate (i.e., ratio between the number of source symbols to the total number of source plus repair symbols) at the sender is fixed and equal to cr. It is initialized after considering the worst receiver experiencing the highest loss rate that should be supported in the session;

From these assumptions we see that ADUs are generated with a fixed period and sent immediately. It forms a convenient time scale, independent of the bw and $symb\_size$ values. Therefore **we measure times in a virtual scale where each ADU generation/transmission corresponding to a "tick"**.

We define a transport protocol added latency, tp_added_lat, using this time scale, as the difference between the time when an ADU is expected at a receiver, assuming no erasure occurred, and the time it is actually delivered to the receiving application. A received ADU is immediately delivered to the application and $tp\_added\_lat = 0$ tick. But

Common	
ADU	Application Data Unit, assumed to contribute to exactly one source symbol in this work
$symb\_size$	symbol size, assumed fixed (in bytes)
bw	application flow bandwidth, assumed fixed (in bps)
$tp\_max\_lat\_in\_sec$	transport protocol maximum latency (in seconds)
tp_max_lat	corresponding transport protocol max. latency (in ticks)
tp_added_lat	transport protocol added latency (in ticks)
cr	AL-FEC coding rate
plr	packet loss rate on the erasure channel
Block codes	
k	block size (in symbols)
Convolutional codes	
ew_size	maximum encoding window size at a sender (in symbols)
dw_size	maximum decoding window size at a receiver (in symbols), or nb of source symbols in L.S. that didn't time-out yet
ls_size	max. linear system size (width) at a receiver (in symbols)
sl	generate a repair symbol after sliding by this number

TABLE I. TERMINOLOGY AND NOTATIONS USED IN THIS WORK.

an erased ADU is only delivered after a certain delay that largely depends on the AL-FEC type, and $tp\_added\_lat > 0$ ticks. The added latency can also be infinite if an ADU could not be recovered on time. When it happens we remove it from latency calculations. Minimizing the average and maximum added latency while keeping the highest possible erasure recovery performance is a key objective.

B. Transport versus Application Added Latency

The added latency at the output of the transport protocol can be different from the added latency seen by the receiving application. Indeed:

- an application may require ADUs to be delivered sequentially, without mis-ordering. In that case an erased ADU delays all the following ADUs until the erasure is recovered or until the ADU times-out;
- an audio/video player is obliged to shift ADU playout by the maximum added latency seen so far (e.g., by using a playout buffer of an adequate size), since content must be consumed in a regular way (we ignore time compression or expansion technics);

These aspects **are not considered** in this work that remains independent of the flow semantic and application.

C. The Case of Block AL-FEC Codes

The maximum latency is directly related to the source block size, k. Indeed, at a receiver, the added latency is maximum when the first source symbol of a block is lost. Recovering from this loss requires to wait for the first repair symbol, sent after the k^{th} packet. Therefore:

$$tp\_max\_lat = k = \frac{tp\_max\_lat\_in\_sec * bw}{8 * symb\_size} \quad (1)$$

This work assumes that all repair packets are sent immediately at the end of the block. If this has a positive impact on the k parameter (larger value) and therefore on robustness, on the downside it also leads to a bursty traffic (periodic bitrate peaks). Future works will consider more realistic CBR transmissions, by spreading repair packet transmission.

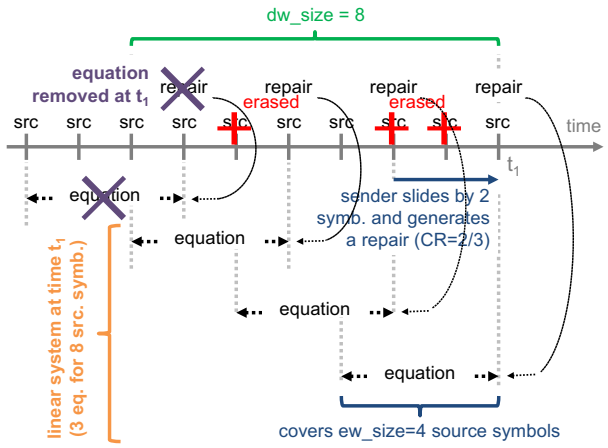


Fig. 2. Convolutional code principles at a receiver, at time t_1 after receiving a new repair symbol.

D. The Case of Convolutional AL-FEC Codes

Two parameters must be considered here (see Fig. 2):

- the maximum encoding window size at a sender, or ew_size (in symbols): this parameter is used during repair symbol creation. For instance, with RLC codes (Section V-A), each repair symbol is a linear combination of ew_size source symbols.
- the maximum decoding window size at a receiver, or dw_size (in symbols): this parameter is the maximum number of received or erased source symbols that can be part of the linear system. Each time the decoding window slides by a certain number of source symbols to the right, the same number of "old" source symbols are removed from the left. An equation of this linear system cannot include source symbols that extend beyond this decoding window, and moving the window to the right can trigger the removal of "old" equations from the linear system;

Here the maximum latency is directly related to the maximum decoding window size, dw_size, since this is the maximum time an erased source symbol can be considered in the linear system. Therefore:

$$tp\_max\_lat = dw\_size = \frac{tp\_max\_lat\_in\_sec * bw}{8 * symb\_size} \quad (2)$$

Given the target code rate cr, the sender will generate a new repair symbol after sliding the encoding window by sl source symbols. Since $cr = \frac{sl}{sl+1}$, we have: $sl = \frac{cr}{1-cr}$.

In a given set of dw_size source symbols, there can be a maximum of $dw\_size - ew\_size$ different encoding windows **fully included** in the set. Therefore the maximum number of repairs in a set of of dw_size source symbols, n_1, is: $n_1 = \frac{dw\_size - ew\_size}{sl}$ repair symbols, or more simply:

$$n_1 = (dw\_size - ew\_size) * (\frac{1}{cr} - 1) \quad (3)$$

This is the maximum number of erasures that could be recovered in any set of dw_size source symbols. However

this result does not take into account the ew_size parameter, i.e., the individual protection brought by a repair symbol. At an extremum, if $ew\_size = 1$, this protection would be rather low in spite of the high number of repair symbols. Therefore we need to also consider the number n_2 of repair symbols that protect a given source symbol in the middle of the set of dw_size source symbols. We have: $n_2 = \frac{ew\_size}{sl}$ or:

$$n_2 = ew\_size * (\frac{1}{cr} - 1) \quad (4)$$

which is also the protection of any source symbol in case of an erasure burst.

In practice protection is limited by the minimum of n_1 and n_2. Since: $min(n_1, n_2) = (\frac{1}{cr} - 1) * min(dw\_size - ew\_size, ew\_size)$, protection is maximum when $min(n_1, n_2)$ is maximum, that is to say when:

$$ew\_size = dw\_size/2 \quad (5)$$

IV. Decoding Beyond Maximum Latency

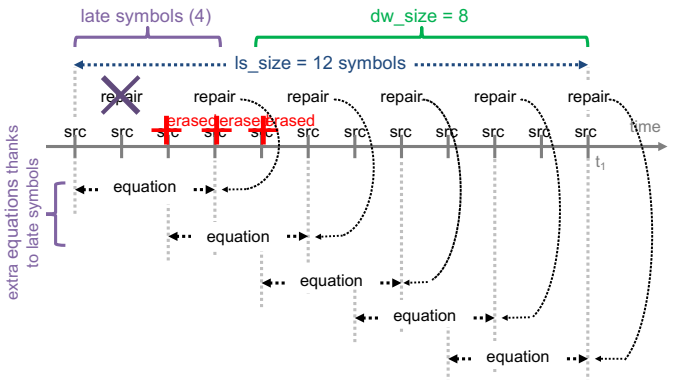


Fig. 3. Late symbols decoding principle, at time t_1.

We now explain how to further improve protection of convolutional codes without impacting maximum latency, at the cost of extra CPU overhead. The idea consists in extending the linear system beyond the decoding window, i.e., having ls_size larger than the value of Eq. 2 (see Fig. 3):

$$ls\_size > dw\_size \quad (6)$$

A source symbol (and ADU) can now be decoded even if the added latency exceeds the maximum value permitted by the application. Therefore these source symbols **must not** be delivered to the application and will be dropped once they are no longer needed. However, we show that decoding these so-called "late symbols" significantly improves the global robustness in bad reception conditions.

Indeed, during a long erasure burst, an equation of size dw_size may be too small to be useful. And if a source symbol cannot be recovered, it impacts several equations, as long as it is part of the sender's encoding window. On the opposite, extending the linear system size beyond dw_size offers a better protection against long erasure bursts: a source symbol that benefits from a late decoding may help decoding more recent source symbols within their validity period. Therefore extending the ls_size value at a receiver can help improving robustness even if it generates additional processing. Finally, this optimization can be set by each receiver independently of the source and other receivers, depending on local criteria only (e.g., an energy vs. robustness target).

To summarize, given the application parameters, Eq. 2 enables to compute dw_size. Then the sender computes $ew\_size = dw\_size/2$, whereas the receiver selects any value for ls_size such that Eq. 6 holds. We will see that $ls\_size = 2 * dw\_size$ (approximately) is a good choice.

V. Erasure Performance Analysis

A. Performance Evaluation Methodology and Choice of Reed-Solomon and RLC Codes

The performance of block and convolutional codes has been assessed through simulated transmissions on a controlled channel, using our OpenFEC (http://openfec.org) AL-FEC implementation and performance evaluation environment. We chose Reed-Solomon as a representative of ideal, MDS block codes, and Random Linear Codes (RLC) as representative of convolutional codes. We only considered Finite Field $GF(2^8)$ and did not try to experiment with sparse variants of RLC in order to only focus on the impacts of the block versus convolutional nature of the codes. We reused our C-language Reed-Solomon codec and derived an RLC codec, both of them relying on the same $GF(2^8)$ library.The `eperftool` evaluation tool of OpenFEC has also been extended to support both types of codes. Actual AL-FEC encodings and decodings are performed using these C-language codecs and various parameters enable to control the simulated channel features. However only memoryless channels are considered in this work, other channels being left to future works. In order to assess decoding throughput, we used a MacbookPro, quadri-core i7/2.5GHz, running MacOS 10.11. Although this powerful laptop is not representative of smartphones, it enables fair comparisons between the two codecs.

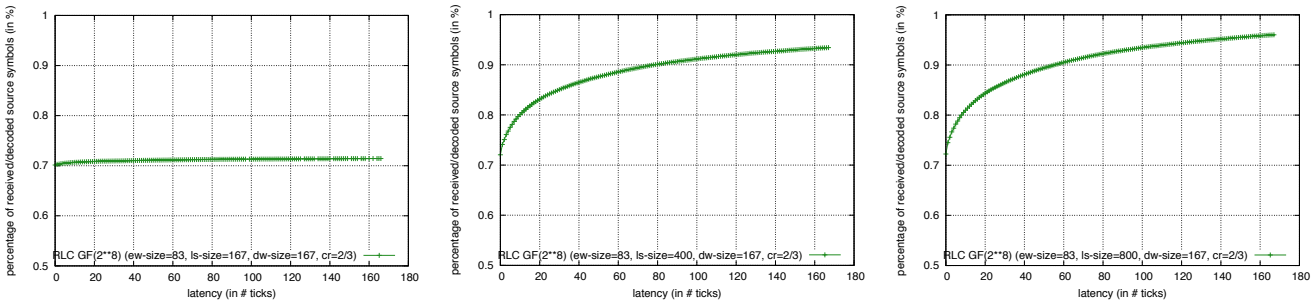
To the best of our knowledge, this environment is unique, and since the same tools and methodology are used, block and convolutional codes can be fairly compared.

B. Benefits of "Decoding Beyond Maximum Latency"

Let us find an appropriate ls_size value. We first evaluate the Cumulative Distribution Function (CDF) for RLC codes as a function of the ls_size parameter chosen by a receiver. Since the benefits are mainly visible when approaching the theoretical decoding limits[1], we consider a loss rate equal to 30%, close to the 33.33% maximum theoretical limit made possible by $CR = 2/3$. Figure 4 demonstrate how efficient this approach is: moving from $ls\_size = 167$ (default solution) to 400 increases the marginal success probability from 71.43% up to 93.42% (let's recall that this success probability ignores packets decoded too late in the CDF computation). Then, values $ls\_size > 400$ do not significantly increase this success probability (96.02% at $ls\_size = 800$).

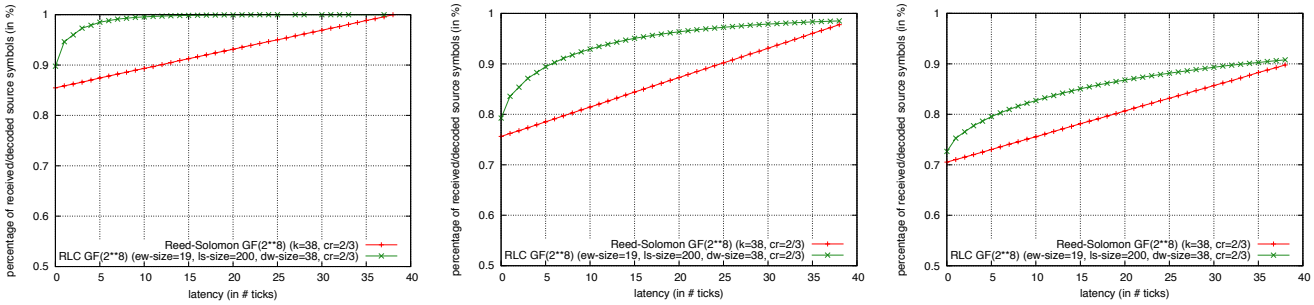
As we expected, Table II shows that increasing the ls_size parameter also increases decoding complexity. Using

[1]In good reception conditions, all erased source symbols are quickly recovered without having to consider late symbol decoding.



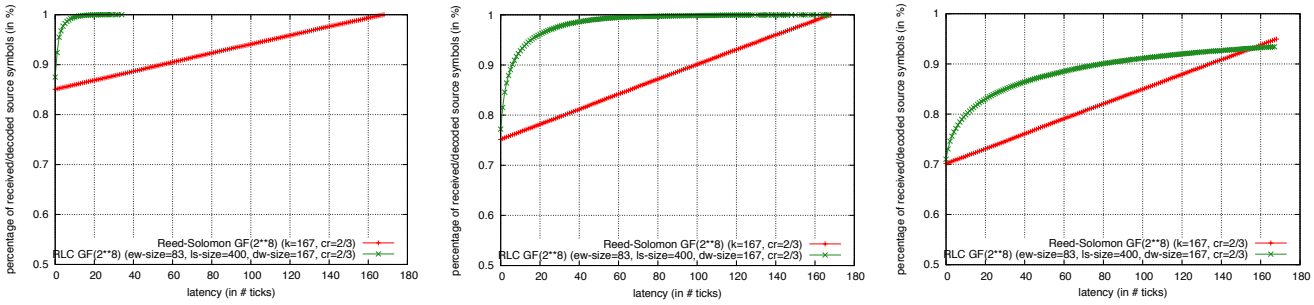
(a) RLC CDF when $ls\_size = 167$ (b) RLC CDF when $ls\_size = 400$ (c) RLC CDF when $ls\_size = 800$

Fig. 4. RLC CDF for received or decoded source packet latency distribution, as a function of ls_size. Here ($ew\_size = 83$, $dw\_size = 167$, $CR = 2/3$).



(a) R-S vs. RLC CDF when $loss = 15\%$ (b) R-S vs. RLC CDF when $loss = 25\%$ (c) R-S vs. RLC CDF when $loss = 30\%$

Fig. 5. R-S and RLC CDF for received or decoded source packet latency distribution, in case of small blocks (38), for various loss rates of a memory-less channel. Comparable parameters are used for both codes, namely $k = 38$ for Reed-Solomon and ($ew\_size = 19$, $ls\_size = 200$, $dw\_size = 38$) for RLC, with $CR = 2/3$ in all cases. A total of $100,000$ source symbols are sent during tests.



(a) R-S vs. RLC CDF when $loss = 15\%$ (b) R-S vs. RLC CDF when $loss = 25\%$ (c) R-S vs. RLC CDF when $loss = 30\%$

Fig. 6. R-S and RLC CDF of received or decoded source packet latency distribution, in case of medium size blocks (167), for various loss rates of a memory-less channel. Comparable parameters are used for both codes, namely $k = 167$ for Reed-Solomon and ($ew\_size = 83$, $ls\_size = 400$, $dw\_size = 167$) for RLC, with $CR = 2/3$ in all cases. A total of $100,000$ source symbols are sent during tests.

ls_size	at 1% loss rate	at 5% loss rate	at 24% loss rate
200	1713.9 Mbps	725.9 Mbps	187.5 Mbps
400	1634.8 Mbps	608.3 Mbps	155.3 Mbps
800	1358.3 Mbps	490.4 Mbps	101.3 Mbps

TABLE II. DECODING SPEEDS FOR RLC CODES AS A FUNCTION OF THE CHANNEL LOSS RATE AND ls_size. HERE ($ew\_size = 83$, $dw\_size = 167$, $CR = 2/3$).

$ls\_size = 400$ appears to be a good balance in terms of decoding speed, when $dw\_size = 167$.

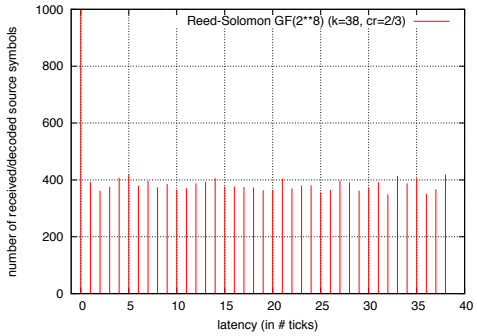
More generally, choosing ls_size roughly equal to twice the dw_size value is considered as a reasonable choice. This rule will be followed throughout the remaining tests.

C. Transport Protocol Added Latency Performance

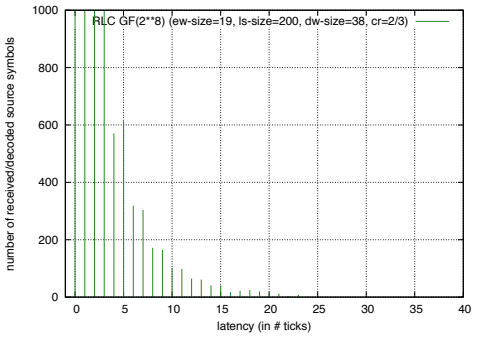
Let us now focus on the transport protocol added latency. Non-surprisingly, Reed-Solomon codes suffer from their block nature. This is visible in the CDF figures for received or decoded source packet latency distribution (Figures 5 and 6). No matter whether we consider a small block ($k = 38$) or a medium size block ($k = 167$), Reed-Solomon CDF curves are significantly below that of RLC.

This difference is easily understood when considering the associated histograms in Figures 7 (truncated to improve readability). With Reed-Solomon ($k = 38$ in that case), a large constant queue distribution remains up to a delay of 38 ticks (block size). On the opposite, with RLC, most ADUs have

been received or decoded in less than 10 ticks.



(a) R-S (truncated, max value is 85472 symbols for latency 0)



(b) RLC (truncated, max value is 89774 symbols for latency 0)

Fig. 7. R-S and RLC histograms of received or decoded source packet latency distribution, in case of small blocks (38), for loss rate 15%.

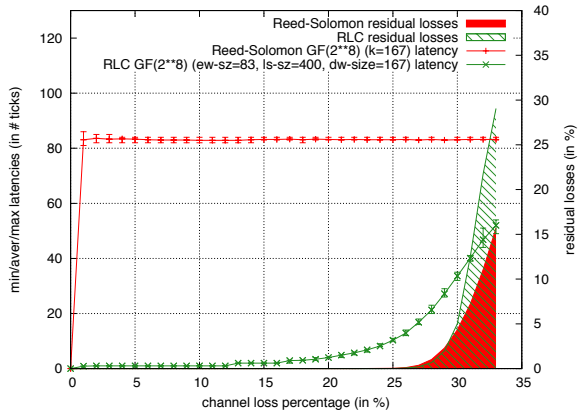


Fig. 8. R-S and RLC added latency and residual loss rate for erased source packets only, as a function of the loss rate of a memory-less channel. Comparable parameters are used for both codes, namely $k = 167$ for R-S and ($ew\_size = 83$; $ls\_size = 400$; $dw\_size = 167$) for RLC, with $CR = 2/3$.

The same phenomenon is highlighted in a different way in Figure 8, where only erased ADUs are considered. This figure shows the FEC decoding latency and the residual loss rates after AL-FEC decoding as a function of the packet loss rate. We see two different behaviors: with block codes, in case of erasures, the first source symbol of a block contributes to a $167-0$ ticks latency once recovered, the second source symbol to a $167 - 1$ ticks latency, etc., till the last symbol of the block that is immediately recovered. On average losses affect all blocks for a packet loss rate as small as 1% since $k = 167 > 100$. Therefore the curve reaches the average $167/2 = 83.5$ ticks latency as soon as $plr = 1\%$ and then remains flat until $plr = 33\%$. The situation is totally different with convolutional codes: the average latency increases very slowly with the plr value. Only the receivers with very bad channels experience a significant FEC-related latency.

Concerning residual losses that could not be recovered on time, we see that both codes behave the same, except for very bad channels (i.e., $plr = 30\%$ and higher) where Reed-Solomon exhibits better results. However we believe that this is not a big issue (e.g., we made some simplification hypotheses that largely favor block codes, see Section III-C).

Globally, we can say that RLC codes exhibit an order of magnitude improvement in terms of FEC-added latency compared to Reed-Solomon codes in most situations.

D. Decoding Speed Performance

Finally we measure the decoding speeds of the RLC and Reed-Solomon codecs[2]. Figures 9 show that if both codecs achieve the same speed when approaching the decoding limit, however RLC exhibits higher speeds in good to medium channel conditions. From this point of view too, there is an incentive to use such convolutional AL-FEC codes as RLC.

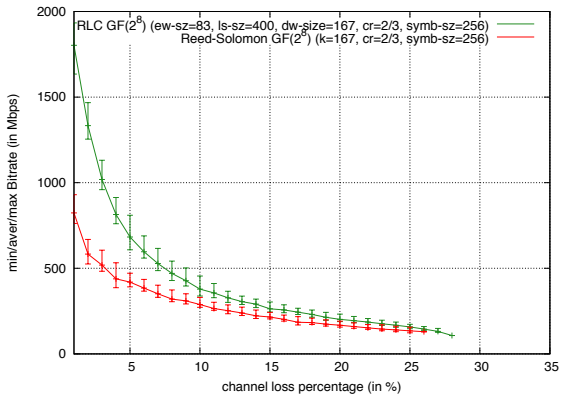


Fig. 9. Reed-Solomon and RLC decoding speed as a function of the loss rate of a memory-less channel. Here $k = 167$ for Reed-Solomon and ($ew\_size = 83$, $ls\_size = 400$, $dw\_size = 167$) for RLC, with $CR = 2/3$ and $symbol\_size = 256$ bytes in all cases.

VI. Related Works

Convolutional AL-FEC codes have received more attention in the academic community in particular for network coding use-cases. Random Linear Network Codes (RLNC) [9] are by nature convolutional codes, designed to accommodate in-transit re-coding operations. In this context, their benefits against block codes has been studied for instance in [10]. Our work differs from several perspectives: we consider broadcast/multicast communications without any feedback channel,

[2]Note that previous results [8] already validated the adequacy of this Reed-Solomon codec for lightweight platforms and more generally RLC techniques [9].

while [10] relies on point-to-point, bidirectional communications; we consider end-to-end (or middlebox to middlebox) protection while [10] considers network coding techniques. Additionally, we explain with great details how to configure these codes and introduce a highly efficient "beyond maximum latency decoding" optimization to convolutional codes.

[11] is a recent work on a similar topic. It introduces a detailed theoretic queuing and coding performance analysis to the problem. Being focused essentially on the analytical aspects of low delay codes, it complements well our work that is more practical, introduces an optimization and another approach for performance evaluation.

[12] proposes to use late decoded packets which is similar to our decoding beyond maximum latency. However here also the work is limited to point-to-point bidirectional communications, where the elastic coding window also evolves according to the acknowledgments obtained from the receiver, and focuses on video flows only. On the opposite, we consider fixed size sliding encoding techniques and carry out performance analyses that are agnostic of the flow nature.

VII. Conclusions and Discussions

This work demonstrates that convolutional AL-FEC codes outperform block codes when dealing with real-time flows: they reduce the FEC-related latency by an order of magnitude while keeping similar erasure recovery performance. It also explains how to initialize convolutional AL-FEC codes in order to comply with real-time constraints, which is less immediate than with block codes. However, following a few simple steps is sufficient. The optimization proposed, whereby each receiver can decide to extend the maximum size of the linear system, significantly reduces the residual losses in very bad channel conditions. Finally using two comparable C-language codecs, it shows that RLC benefits from improved decoding speeds in good to medium quality channels, which immediately translates into reduced power consumption, an important feature with lightweight terminals. All these results motivate our work on extending FECFRAME (RFC 6363) to convolutional codes [13], [14].

Future works will consider other channel loss models, beyond the memoryless channels we used here. Future works will also consider the case of constant bitrate (CBR) communications at the output of the FECFRAME sender. In that case the application incoming bitrate needs to be adjusted accordingly in order to keep the output bitrate constant. This scenario is representative of use-cases where CBR channels are used (e.g., with 3GPP MBMS services) or situations where adding robustness should not increase the risk of creating congestion within the network. Preliminary results suggest that convolutional code benefits are even more obvious, in particuly in terms of improved robustness (in addition to reduced latency), which we could not exhibit here because of our assumptions.

References

[1] T. Paila, R. Walsh, M. Luby, V. Roca, and R. Lehtonen, "FLUTE - File Delivery over Unidirectional Transport," Nov. 2012, IETF Request for Comments, RFC 6726.

[2] M. Watson, A. Begen, and V. Roca, "Forward error correction (fec) framework," Jun. 2011, IETF Request for Comments, RFC 6363.

[3] "3gpp; technical specification group services and system aspects; multimedia broadcast/multicast service (mbms); protocols and codecs (release 13)," Mar. 2016, 3GPP TR 26.346 version 13.4.0 Release 13.

[4] M. Luby, A. Shokrollahi, M. Watson, and T. Stockhammer, "Raptor Forward Error Correction Scheme for Object Delivery," Oct. 2007, IETF Request for Comments, RFC 5053 (Standards Track).

[5] M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, and L. Minder, "RaptorQ Forward Error Correction Scheme for Object Delivery," IETF Request for Comments, RFC 6330 (Standards Track), Aug. 2011.

[6] J. Lacan, V. Roca, J. Peltotalo, and S. Peltotalo, "Reed-solomon forward error correction (fec) schemes," Apr. 2009, IETF Request for Comments RFC 5510.

[7] V. Roca, C. Neumann, and D. Furodet, "Low density parity check (ldpc) staircase and triangle forward error correction (fec) schemes," Jun. 2008, IETF Request for Comments, RFC 5170.

[8] V. Roca, M. Cunche, C. Thienot, J. Detchart, and J. Lacan, "RS + LDPC-Staircase codes for the erasure channel: Standards, usage and performance," in *9th IEEE Conf. on Wireless and Mobile Computing, Networking and Communications (WiMob 2013)*, Aug. 2013.

[9] F. Fitzek, M. Pedersen, J. Heide, and M. Medard, "Network coding: Applications and implementations on mobile devices," in *5th ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks*, Oct. 2010.

[10] M. Toemoeskoezi, F. Fitzek, D. Lucani, M. Pedersen, and P. Seeling, "On the delay characteristics for point-to-point links using random linear network coding with on-the-fly coding capabilities," in *20th European Wireless Conference*, May 2014.

[11] M. Karzand, D. Leith, J. Cloud, and M. Medard, "Fec for lower in-order delivery delay in packet networks," in *arXiv:1509.00167v2*, Sep. 2016.

[12] P.-U. Tournoux, T. Tran-Thai, E. Lochin, and J. Lacan, "When on-the-fly erasure code makes late video decoding happen," in *25th ACM Workshop on Network and OS Support for Digital Audio and Video (NOSSDAV'15)*, Mar. 2015.

[13] V. Roca and A. Begen, "Forward error correction (fec) framework version 2," Feb. 2017, IETF TSVWG work in progress, draft-roca-tsvwg-fecframev2-03. [Online]. Available: https://datatracker.ietf.org/doc/draft-roca-tsvwg-fecframev2/

[14] V. Roca, "Random linear codes (rlc) forward error correction (fec) scheme for fecframe," Feb. 2017, IETF TSVWG work in progress, draft-roca-tsvwg-rlc-fec-scheme-00. [Online]. Available: https://datatracker.ietf.org/doc/draft-roca-tsvwg-rlc-fec-scheme/